

HP-CAST, November 15, 2014

Dr. Ronald Young

What Will Limit Performance When Floating Point Operations are Free?

HP SL270 Server with (8) NVIDIA K40m GPUs – [Matrix]Warrior benchmark

Performance (Gflops)

Routine	All	CPU's	GPU 0	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7
Matrix Multiply CPU(0%)	9430	0	1197	1201	1188	1210	1203	1206	1181	1185

Machine	Year	Flops	\$/Gflop
DEC VAX	1978	97,000	\$2,000,000,000
(8) NVIDIA GPUs	2014	9,430,000,000,000	\$10
Ratio		~100,000,000	~200,000,000

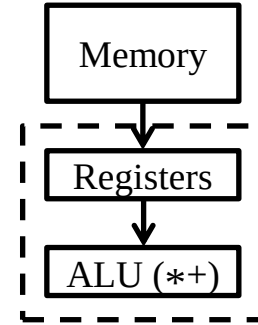
Computer Evolution



Early Machines

Limited by Floating Point & Storage

Processing speed = Memory speed



Computer Evolution



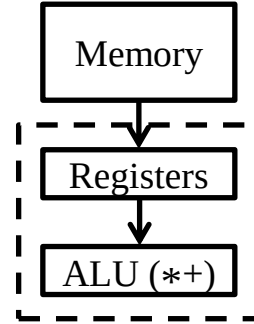
Early Machines

Limited by Floating Point & Storage

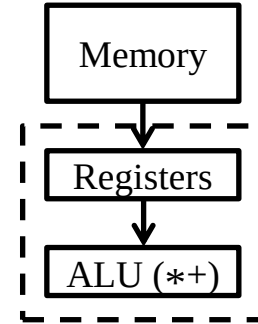
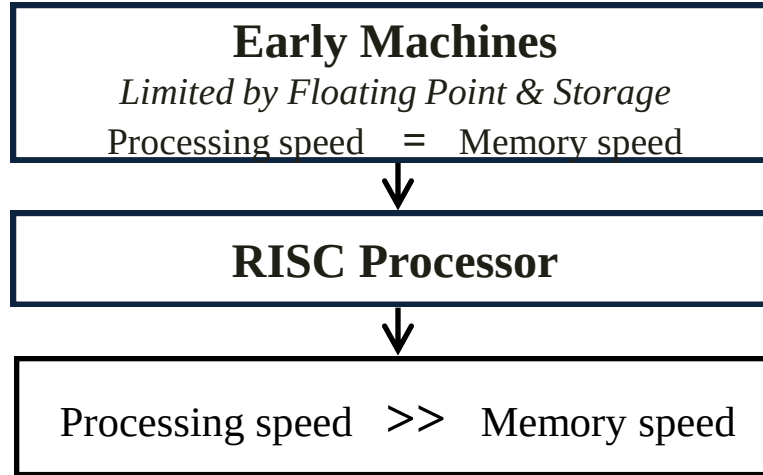
Processing speed = Memory speed



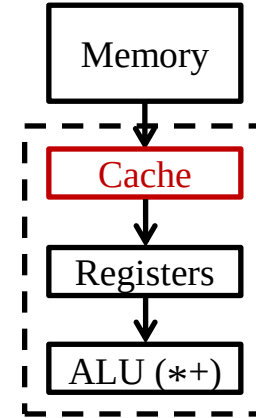
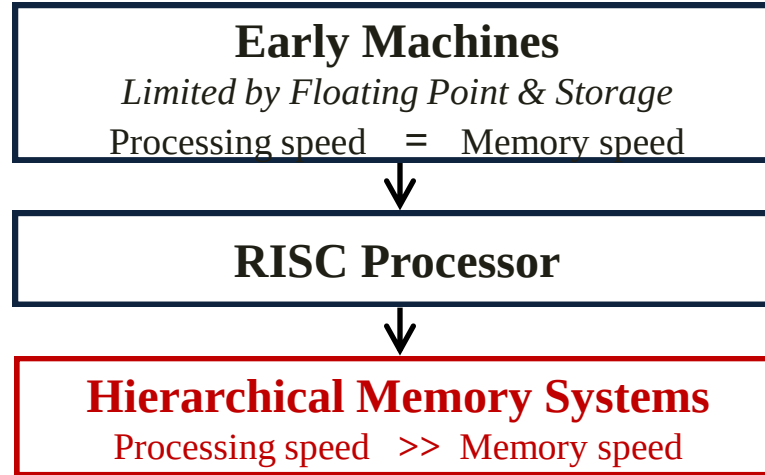
RISC Processor



Computer Evolution



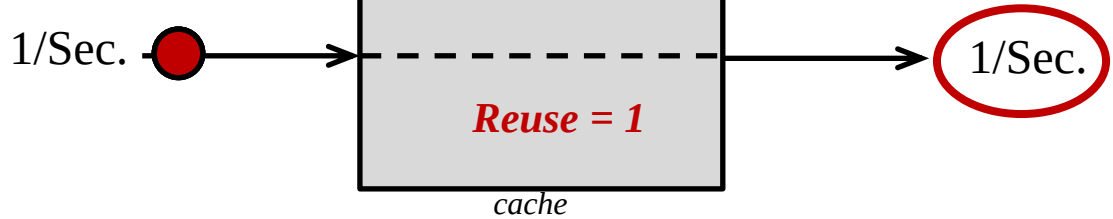
Computer Evolution



Reuse *Efficiently linking slower input to faster requirement*

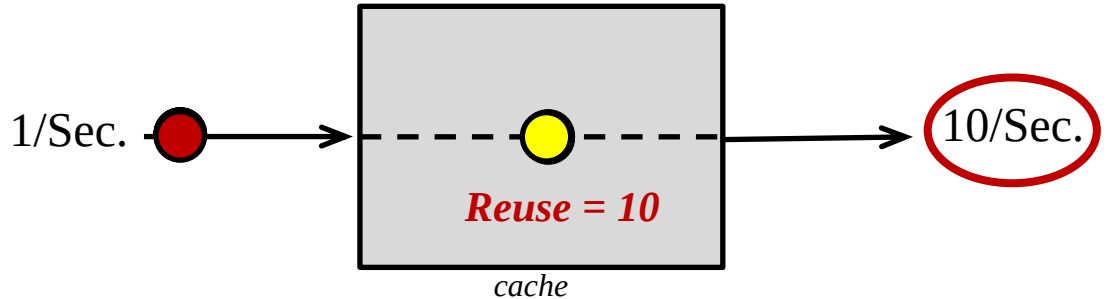
- Reuse = Number of times a data element is used in a computation.*

Old machines: Low Reuse

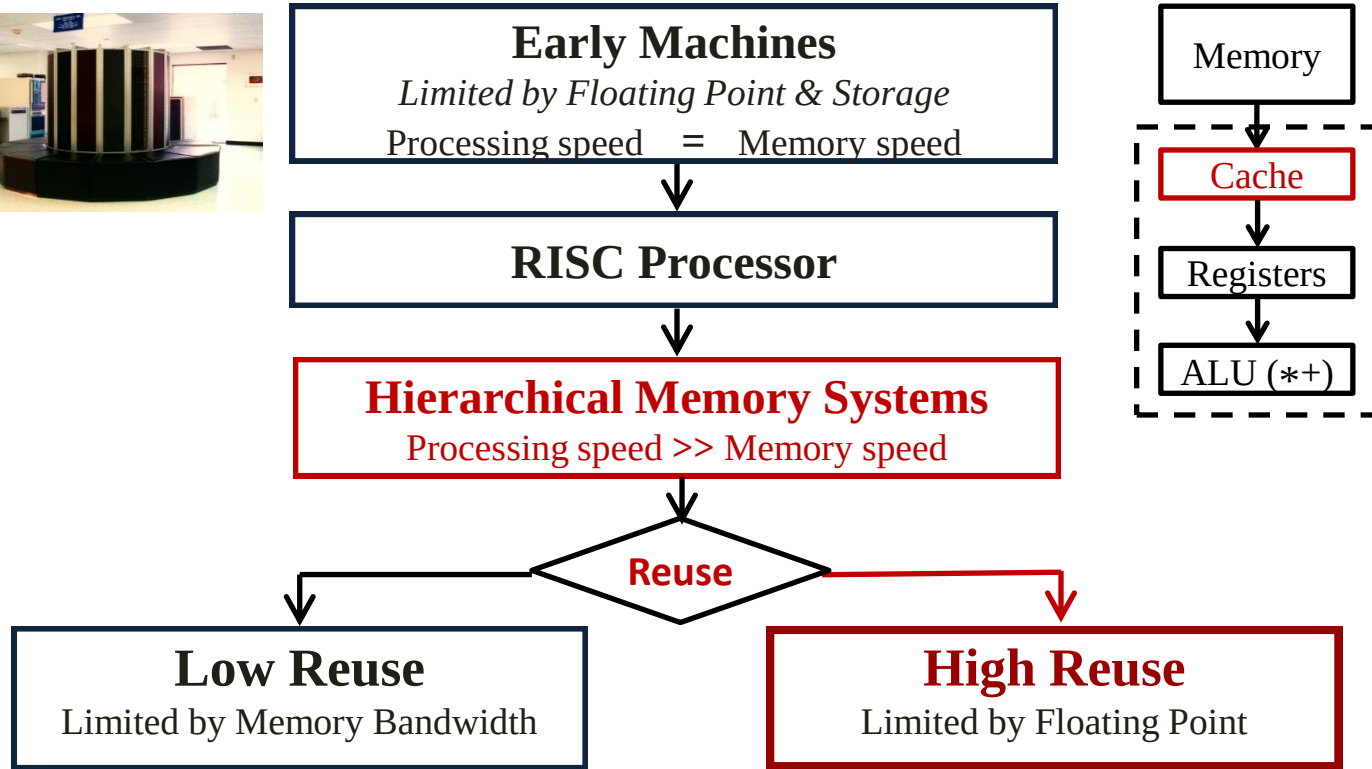


New machines: **High Reuse**

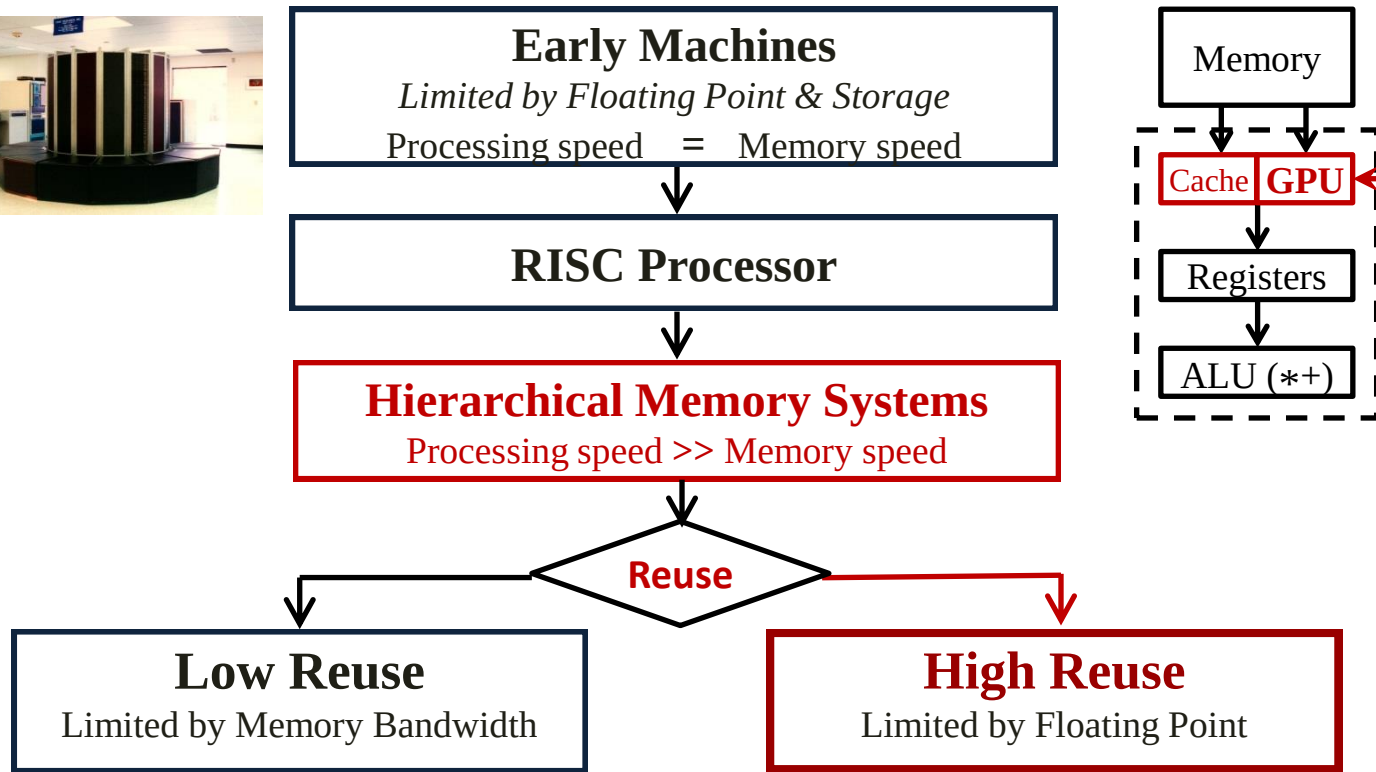
Just an example: can be much higher than 10



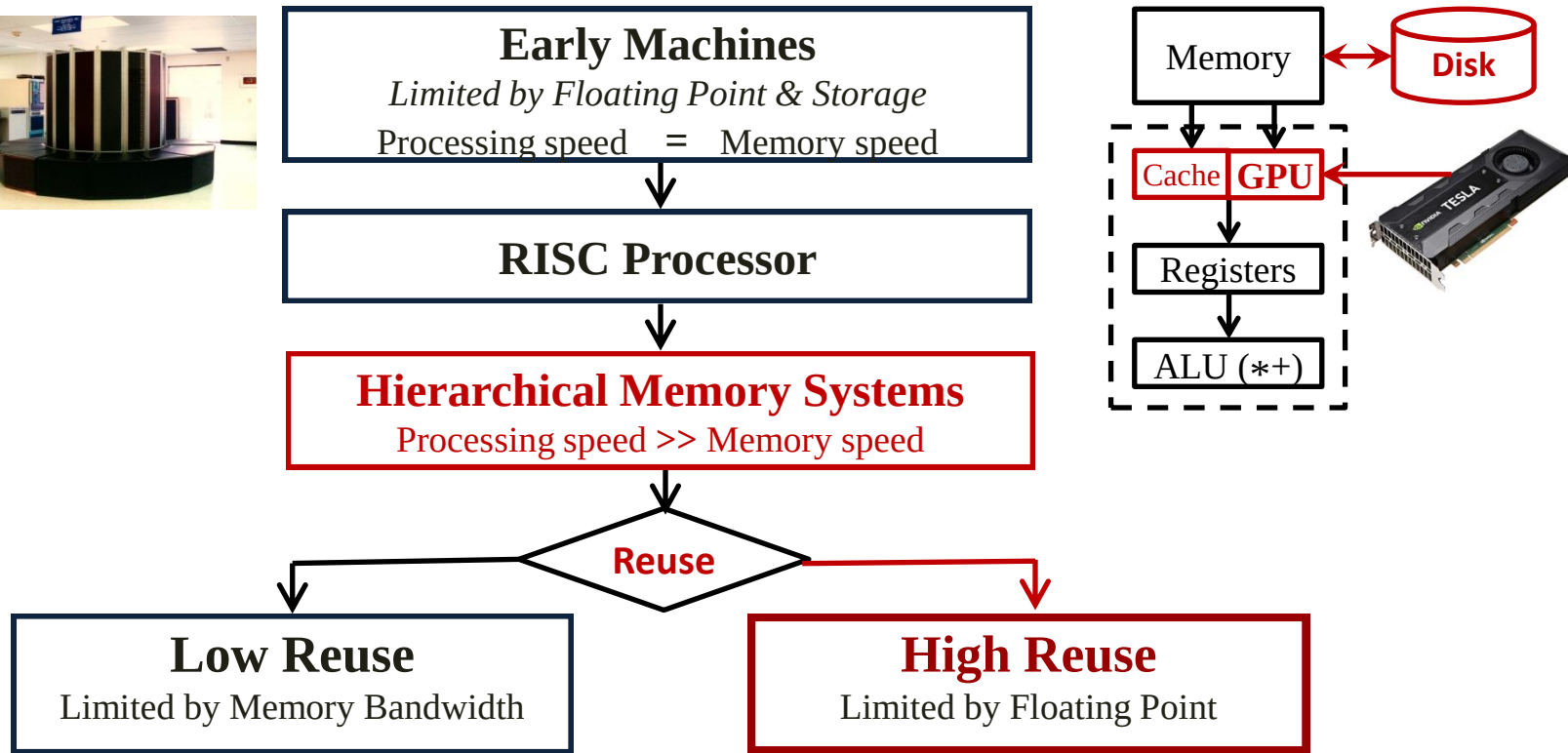
Computer Evolution



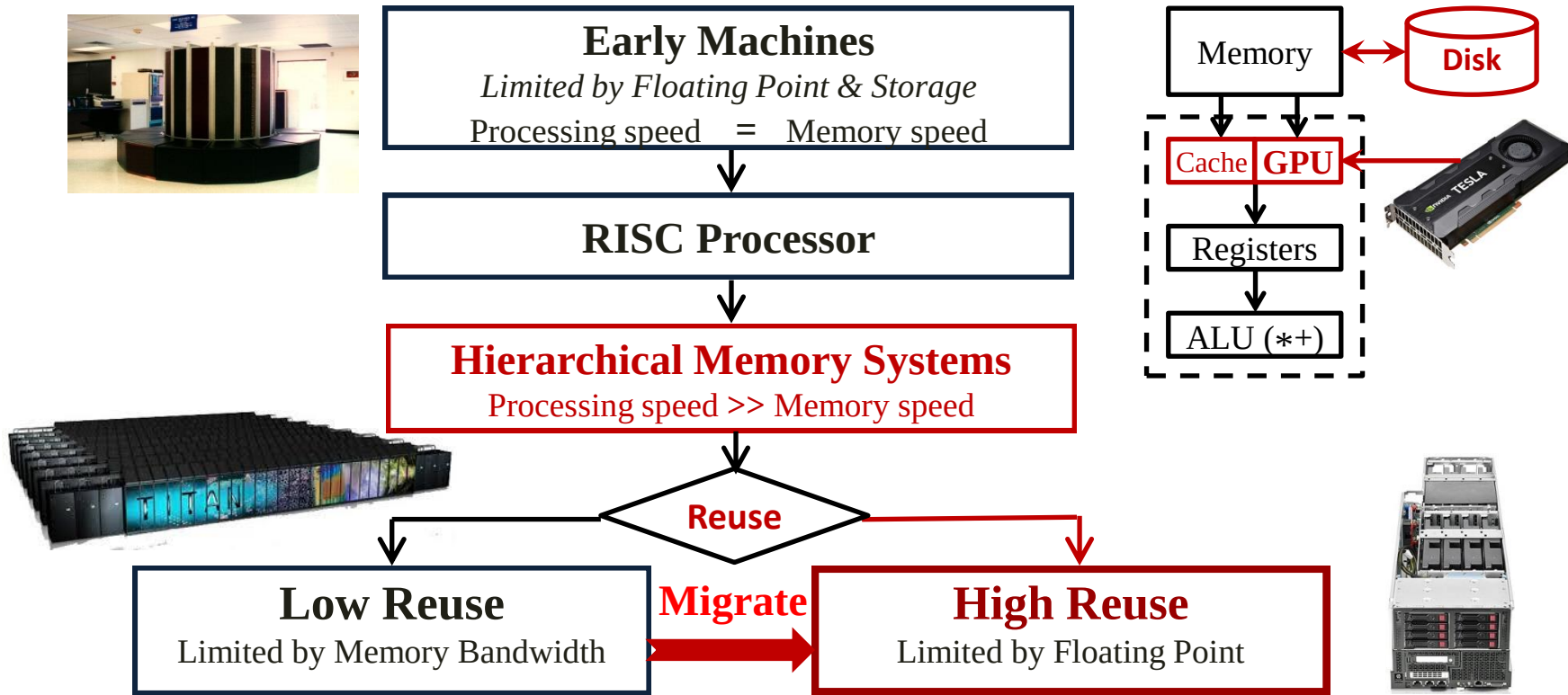
Computer Evolution



Computer Evolution



Computer Evolution



BLAS 1 Reuse

When $M=N=K$

$$\begin{bmatrix} \circ C_{ij} \end{bmatrix}_{C(N,N)} = \begin{bmatrix} \text{---} \end{bmatrix}_{A(N,N)} \begin{bmatrix} | \end{bmatrix}_{B(N,N)}$$

```
DO 30 J=1,N
  DO 20 I=1,M
    DO 10 L=1,K
      C(I,J) = A(I,L)*B(L,J)
    10 CONTINUE
  20 CONTINUE
30 CONTINUE
```

BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1

$$\frac{\text{Flops}}{\text{Data}} = \frac{2N}{(2N+1)} \sim 1$$

BLAS 2 Reuse

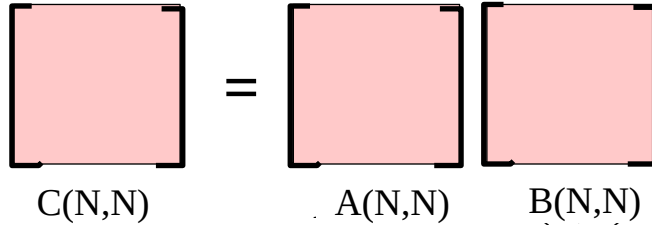
$$\begin{bmatrix} | \\ \hline \end{bmatrix}_{C(N,N)} = \begin{bmatrix} \square \\ \hline \end{bmatrix}_{A(N,N)} \begin{bmatrix} | \\ \hline \end{bmatrix}_{B(N,N)}$$

```
DO 30 J=1,N  
  DO 20 I=1,M  
    DO 10 L=1,K  
      C(I,J) = A(I,L)*B(L,J)  
    10 CONTINUE  
  20 CONTINUE  
30 CONTINUE
```

BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1
2	DGEMV	N^2+2N	$2N^2$	2

$$\frac{\text{Flops}}{\text{Data}} = \frac{2N^2}{(N^2+2N)} \sim 2$$

BLAS 3 Reuse



```
DO 30 J=1,N
  DO 20 I=1,M
    DO 10 L=1,K
      C(I,J) = A(I,L) * B(L,J)
10    CONTINUE
20  CONTINUE
30  CONTINUE
```

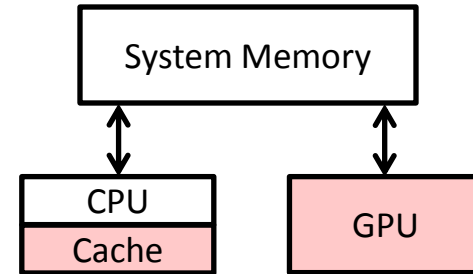
BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1
2	DGEMV	N^2+2N	$2N^2$	2
3	DGEMM	$3N^2$	$2N^3$	$2N/3$

$$\frac{\text{Flops}}{\text{Data}} = \frac{2N^3}{3N^2} = \frac{2N}{3}$$

Reuse – Small Data

$$\begin{bmatrix} \text{[red square]} \end{bmatrix}_{C(N,N)} = \begin{bmatrix} \text{[red square]} \end{bmatrix}_{A(N,N)} \begin{bmatrix} \text{[red square]} \end{bmatrix}_{B(N,N)}$$

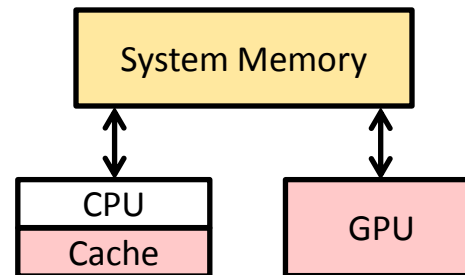
BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1
2	DGEMV	N^2+2N	$2N^2$	2
3	DGEMM	$3N^2$	$2N^3$	$2N/3$



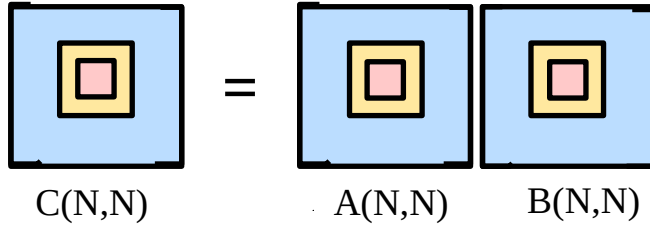
Reuse – More Data

$$\begin{bmatrix} \boxed{\boxed{\text{C(N,N)}}} \end{bmatrix} = \begin{bmatrix} \boxed{\boxed{\text{A(N,N)}}} \end{bmatrix} \begin{bmatrix} \boxed{\boxed{\text{B(N,N)}}} \end{bmatrix}$$

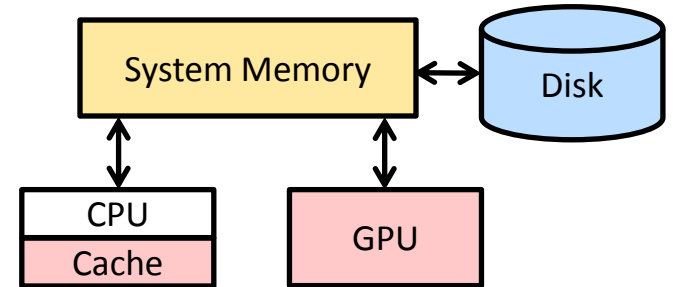
BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1
2	DGEMV	N^2+2N	$2N^2$	2
3	DGEMM	$3N^2$	$2N^3$	$2N/3$



Reuse – Big Data

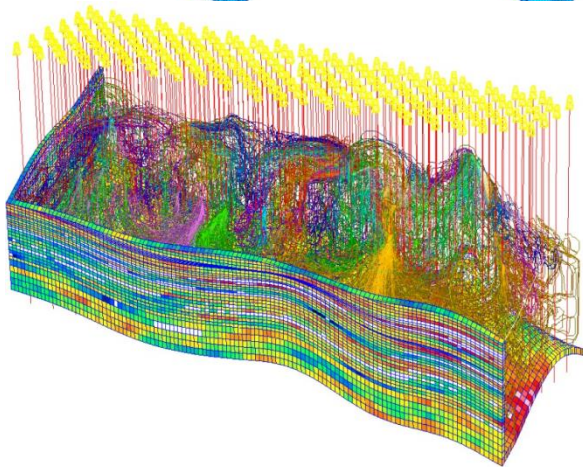
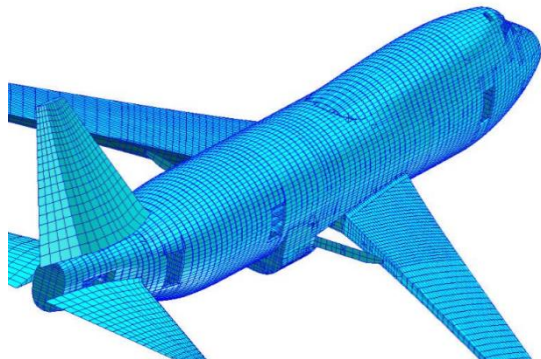


BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1
2	DGEMV	N^2+2N	$2N^2$	2
3	DGEMM	$3N^2$	$2N^3$	$2N/3$



Application Reuse

Discretization



1. Domain is divided into a large number of pieces.
2. Equations are formed for each piece $\{a\}, \{b\}$
3. Pieces are assembled into a global system of equations

$$[A]\{X\} = \{B\}$$

4. The system is then solved for the solution $\{X\}$

More pieces gives better answer

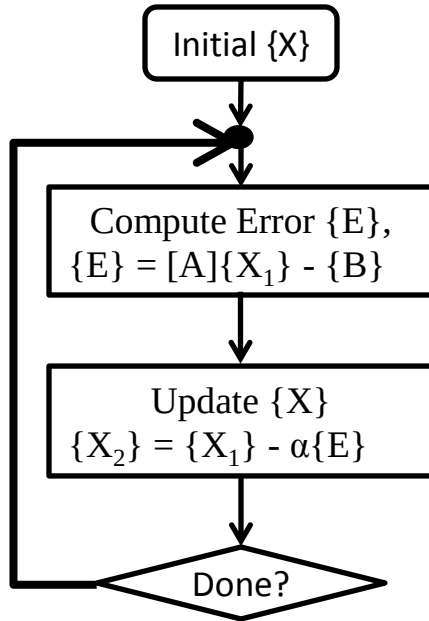
More pieces = more data and processing

$\therefore$ Machine (speed, storage) determines accuracy

Low reuse

Solving $[A]\{X\}=\{B\}$

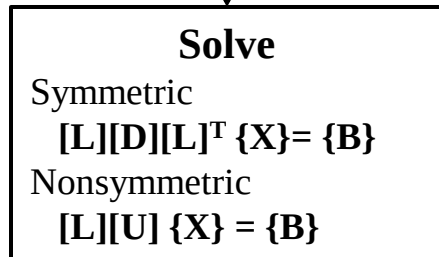
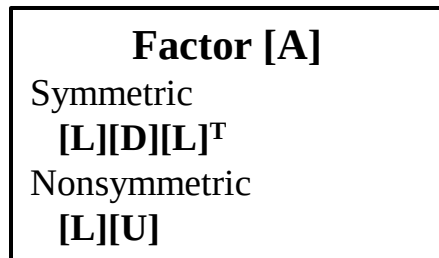
Explicitly



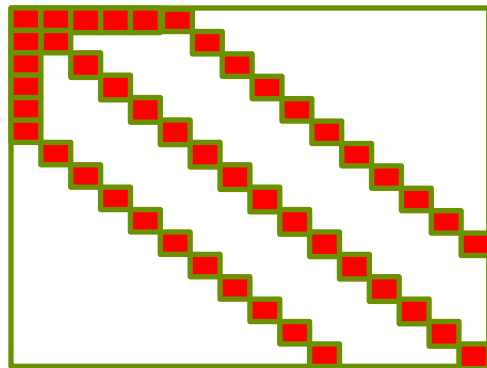
The diagram visualizes the equation $\{E\} = [A]\{X\} - \{B\}$ using matrix representations. On the left is a vertical column of blue squares representing the error vector $\{E\}$. This is followed by an equals sign. In the center is a square matrix of red squares representing the coefficient matrix $[A]$, which has a banded structure. To the right of the matrix is another vertical column of blue squares representing the solution vector $\{X\}$. This is followed by a minus sign and a final vertical column of red squares representing the right-hand side vector $\{B\}$.

- (+) Minimum data
- (-) Low reuse: BLAS 2
- (-) Indirect addressing
- (-) Limited by memory size
- (-) Many iterations
- (-) Convergence risk
- (-) Data dependent
- (-) Unpredictable run times
- (-) Multiple solutions
- (-) Stability -> small increments

High reuse



Solving $[A]\{X\}=\{B\}$



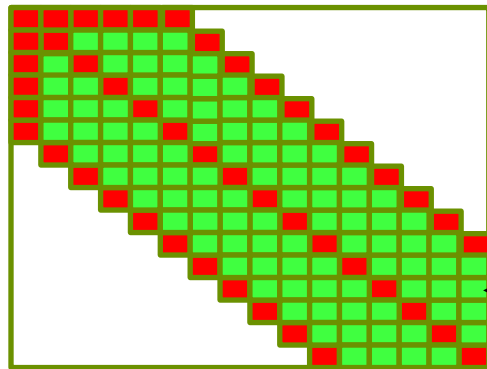
$[A]$



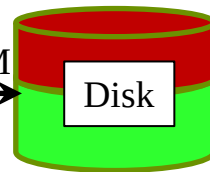
=



$\{X\} \quad \{B\}$



WORM

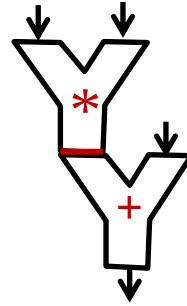
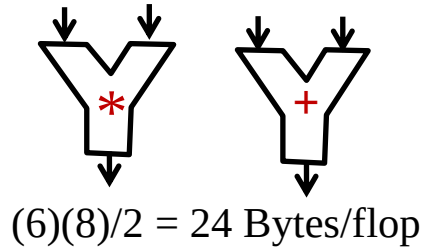


Solid State Disk (SSD)
and
Shingled Magnetic Rec.(SMR)

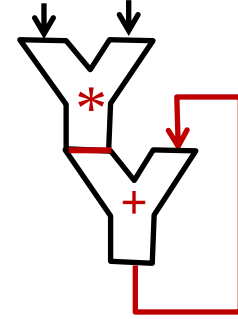
Implicitly

- (-) Fill data
- (+) High reuse: BLAS 3
- (+) Incremental addressing
- (+) One iteration
- (+) No convergence risk
- (+) Data value independent
- (+) Predictable run times
- (+) Multiple solutions
- (+) Unconditionally stable

Hardware Reuse



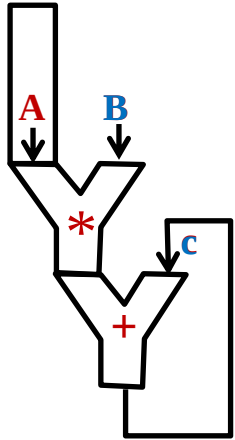
Fused Multiply-Add
2008: IEEE 754
 $(4)(8)/2 = 16 \text{ Bytes/flop}$



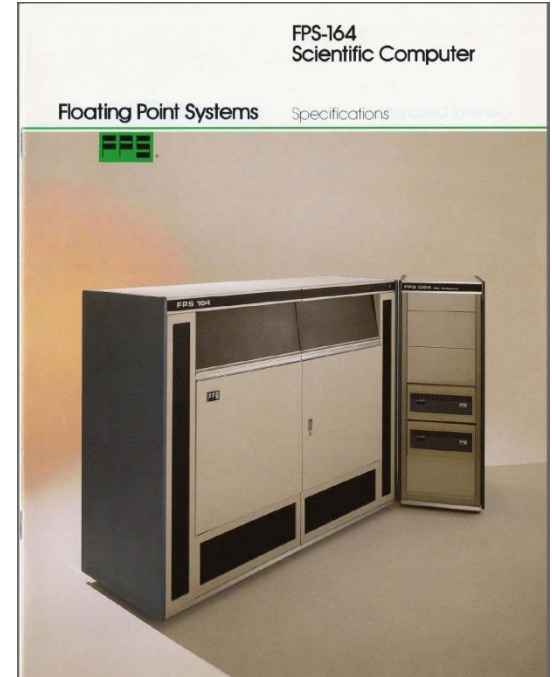
Dot Product
 $(2)(8)/2 = 8 \text{ Bytes/flop}$

Hardware Reuse

$$\begin{bmatrix} \text{O} \\ \text{c} \end{bmatrix} = \begin{bmatrix} \text{---} \\ \text{{A}} \end{bmatrix} \begin{bmatrix} \text{---} \\ \text{{B}} \end{bmatrix}$$



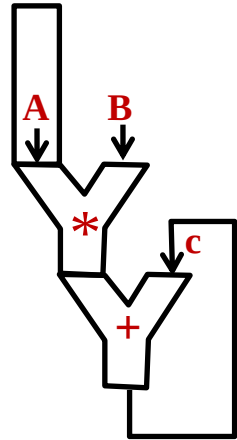
Vector Dot Product
(1)(8)/2 = 4 Bytes/flop



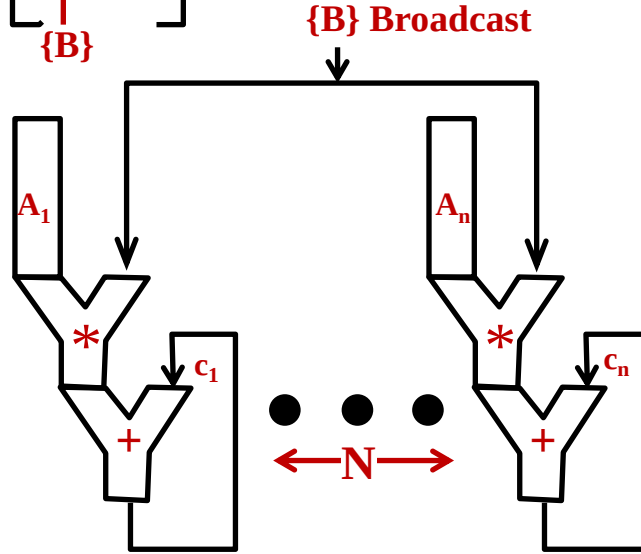
Floating Point Systems 164
First 64-bit accelerator
First version of FMSlib

Hardware Reuse

$$\begin{bmatrix} \textcircled{c_1} \\ \vdots \\ \textcircled{c_n} \\ c \end{bmatrix} = \begin{bmatrix} \{A_1\} \\ \vdots \\ \{A_n\} \end{bmatrix} \begin{bmatrix} | \\ | \\ | \\ \{B\} \end{bmatrix}$$



Vector Dot Product
(1)(8)/2 = 4 Bytes/flop



Parallel Vector Dot Product
(1/N)(4) = (4/N) Bytes/flop

THREE HUNDRED FORTY ONE MILLION FLOATING POINT OPERATIONS PER SECOND.

THE FPS-164/MAX.

Today's scientific and engineering problems are reaching out for ever-complex models and higher resolution, which leads to calculations on very large matrices. The world now looks to a super computer with the speed and accuracy needed to solve these problems before out of reach for many.

Now, there's the FPS-164/MAX—a special purpose, modular supercomputer that matches the likes of CRAY, CYBER, and others in computing speed and accuracy—as a fraction of the cost.

The FPS-164/MAX is fast.

Its peak performance is over 35 to 180 million floating point operations per second, depending on configuration, and up to 7 times that of 88 bit memory available. In fact, since the supercomputer you need to solve these massive computational problems. The FPS-164/MAX configuration is able to compute up to 64 vector elements at one time, allowing a fully configured 884/MAX to factor a 1,000 to 1,000 matrix in about 3 seconds, multiply two 5,000 by 5,000 matrices in less than two hours.

FPS-164/MAX Specifications

Peak Speed for (FP32/FP16)	35
Number of Arithmetic Operations	40
Number of Interconnected Processors	20 x 10
Vector Register Capacity	20 x 10
State Memory Capacity	2 GB
State Subsystem Capacity	1 GB
Word Size	64 bit
Memory	16 GB
Memory Transfer Rate	1.5 x 10 ⁹ bytes/sec
Operating System	UNIX
Language	FORTRAN, C, Pascal, etc.
Weight	1,000 lbs.
Power	1000 watts

The FPS-164/MAX is powerful.

A single processor module is designed to run at 100MHz or higher speed. The FPS-164/MAX has all the other capabilities of our original FPS-164. We've just added a lot more power with multiple special processing units which simplify the vector processing capability of the original FPS-164 by up to 32 times.

The FPS-164/MAX is cost-effective.

In research, analysis, and industrial chemistry and physics, fluid flow analysis, electromagnetic modeling, or any application requiring the handling of large matrices, the FPS-164/MAX offers unparalleled cost efficiency. In fact, if you can't solve key matrix computations as fast or faster than supercomputers costing over \$100,000 a month.

Whether you're looking to replace your existing FPS-164—or upgrading for a complete new system—you will find supercomputer performance for the cost of a day or less in your own lab.

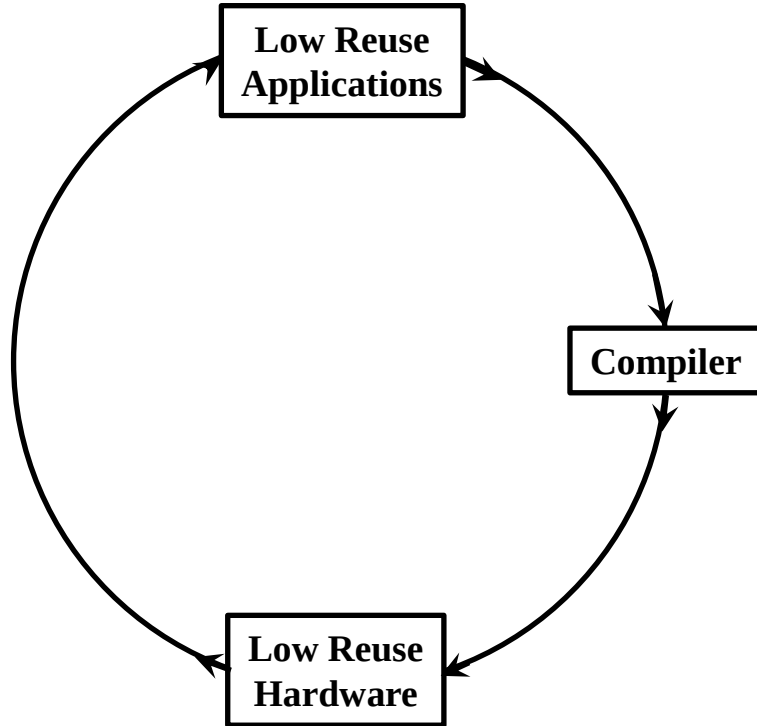
What's more, the FPS-164/MAX is tailored for the immediate expansion of Floating Point Systems. With 20 local service offices world-wide, full system diagnostic capabilities, and a record of product quality and reliability second to none, you can be sure the FPS-164/MAX will be up, running, and ready to solve your problem, making results.

For complete information and quotations, call toll free 1-800-527-1445.

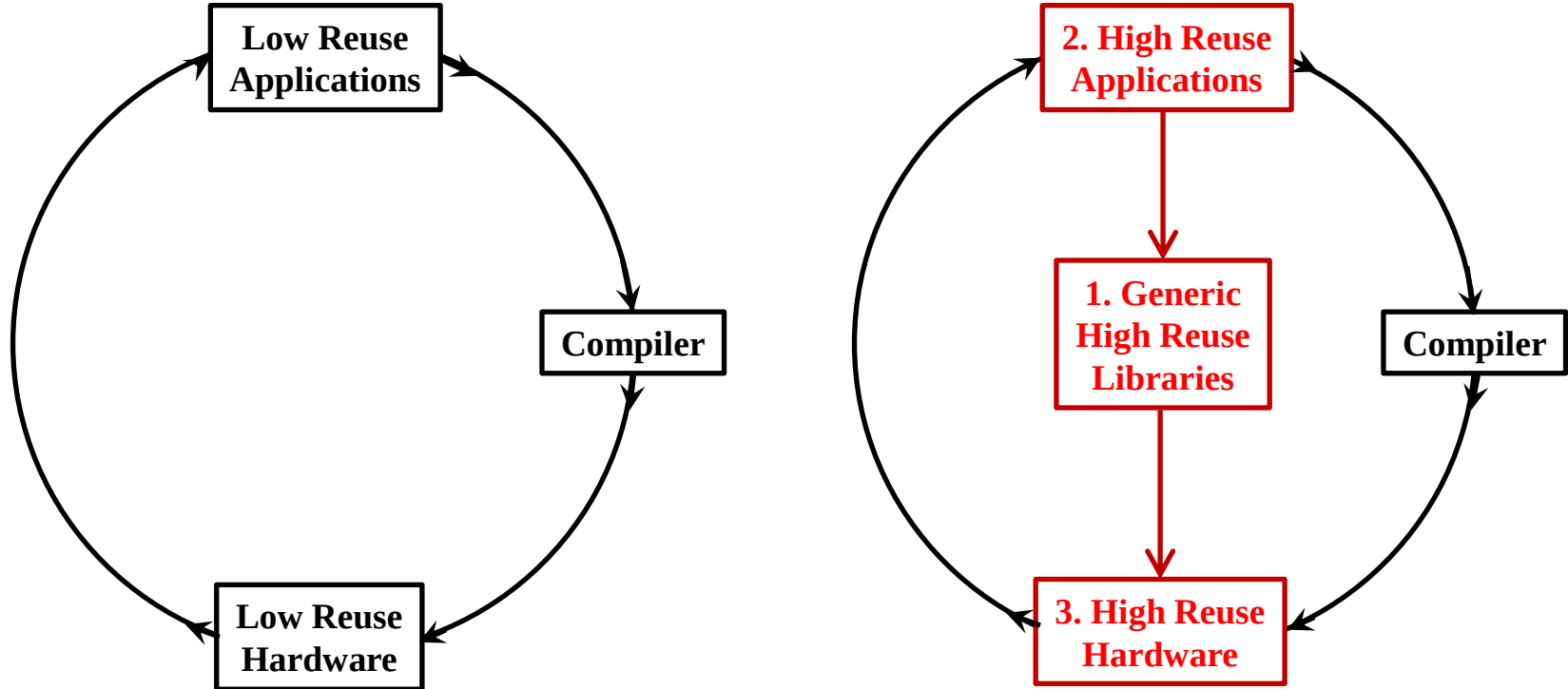
FMSlib Accelerator

Circle Number 333 on Reader Service Card

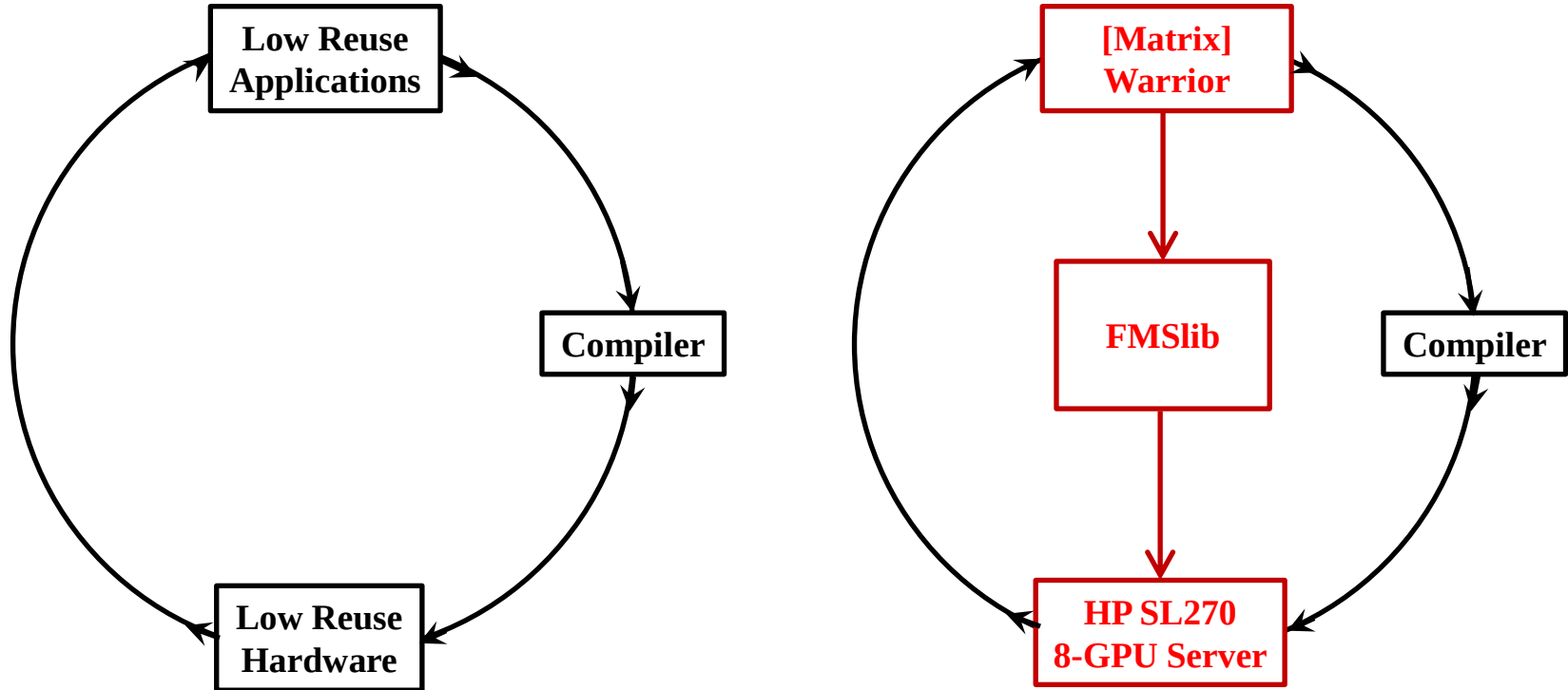
Migrating to **High Reuse**



Migrating to **High Reuse**



Migrating to **High Reuse**



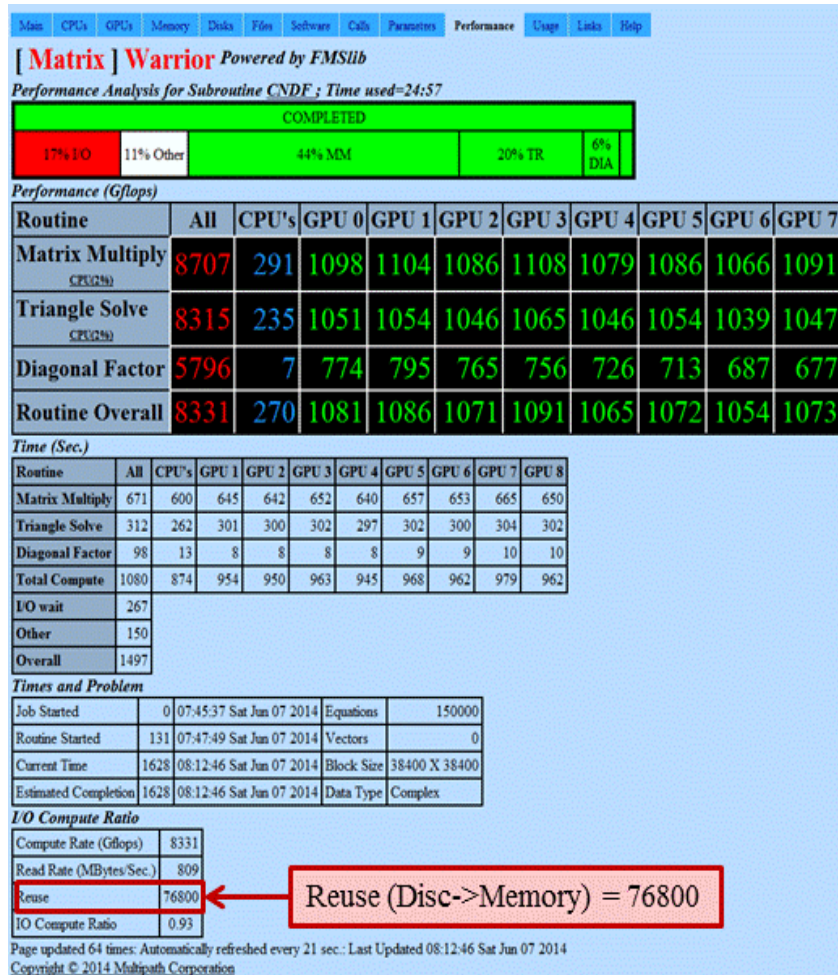
Explore Reuse

Download and run [Matrix]Warrior, an application which benchmarks and demonstrates FMSlib

[Matrix]Warrior

FMSlib

- Form $[A]$, $\{B\}$ with test data
- Use FMSlib to solve $[A]\{X\}=\{B\}$
- Display reports as WEB pages



**Q: What Will Limit Performance
*When Floating Point Operations are Free?***

1. Compiled low reuse software
2. Lack of generic high reuse libraries
3. Lack of high reuse hardware

More Information:

fmslib.com

FREE Download
[Matrix]Warrior

This Presentation

NVIDIA GTC Presentations

Fast Matrix Solver



World's fastest solutions for

$$[A]\{X\} = \{B\}$$

DESCRIPTION

WHY FMS?

WHY GPUS?

WHY DISKS?

APPLICATIONS

BENEFITS

FEATURES

EASE OF USE

BENCHMARKS

MANUAL

CONTACT

DOWNLOADS

FREE: [MATRIX] WARRIOR

FMSLIB

PRESENTATIONS

HP-CAST

Nov 15, 2015

What Will Limit Performance When Floating Point Operations are Free?

NVIDIA GTC Technology Conference

Mar 21, 2013

Benchmark your GPUs with MatrixWarrior

May 17, 2012

Teraflop GPU Acceleration Of Large Matrix Algebra